In [1]:

```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.optim import lr_scheduler
from torch.autograd import Variable
from torch.utils.data import DataLoader
from torch.utils.data import sampler
import importlib
import time

import os
import os.path as osp
import copy

import torchvision.datasets as dset
import torchvision.transforms as transforms
import torchvision.models as models

import numpy as np

import timeit
#import customDataset
#from customDataset import CustomDataset
from importlib import reload


# for google colab runs
!unzip "vgg-train"
```

```
Archive:  vgg-train.zip
replace vgg-train/test/1/00010257.jpg? [y]es, [n]o, [A]ll, [N]one, [r]ename:
N
```

In [2]:

```
class ChunkSampler(sampler.Sampler):
    """Samples elements sequentially from some offset.
    Arguments:
        num_samples: # of desired datapoints
        start: offset where we should start selecting from
    """
    def __init__(self, num_samples, start = 0):
        self.num_samples = num_samples
        self.start = start

    def __iter__(self):
        return iter(range(self.start, self.start + self.num_samples))

    def __len__(self):
        return self.num_samples

use_gpu = torch.cuda.is_available()
if use_gpu:
    print("Using CUDA")
```

```
Using CUDA
```

In [3]:

```
data_dir = 'vgg-train'
TRAIN = 'train'
VAL = 'val'
TEST = 'test'

data_transforms = {
    TRAIN: transforms.Compose([
        transforms.CenterCrop(224),
        transforms.RandomHorizontalFlip(),
        transforms.ToTensor(),
    ]),
    VAL: transforms.Compose([
        transforms.Resize(256),
        transforms.CenterCrop(224),
        transforms.ToTensor(),
    ]),
    TEST: transforms.Compose([
        transforms.Resize(256),
        transforms.CenterCrop(224),
        transforms.ToTensor(),
    ])
}

image_datasets = {
    x: dset.ImageFolder(
        osp.join(data_dir, x),
        transform=data_transforms[x]
    )
    for x in [TRAIN, VAL, TEST]
}

dataloaders = {
    x: torch.utils.data.DataLoader(
        image_datasets[x], batch_size=16,
        shuffle=True
    )
    for x in [TRAIN, VAL, TEST]
}

dataset_sizes = {x: len(image_datasets[x]) for x in [TRAIN, VAL, TEST]}

for x in [TRAIN, VAL, TEST]:
    print("Loaded {} images under {}".format(dataset_sizes[x], x))

print("Classes: ")
class_names = image_datasets[TRAIN].classes
print(image_datasets[TRAIN].classes)
```

```
Loaded 294 images under train
Loaded 125 images under val
Loaded 125 images under test
Classes:
['0', '1', '2', '3', '4', '5']
```

In [0]:

```
class Flatten(nn.Module):
    def forward(self, x):
        N, C, H, W = x.size() # read in N, C, H, W
        return x.view(N, -1)  # "flatten" the C * H * W values into a single vector per image
```

```
In [0]: def train_model(vgg, criterion, optimizer, scheduler, num_epochs=10, save=Fals
e, save_filename=""):
    since = time.time()
    best_model_wts = copy.deepcopy(vgg.state_dict())
    best_acc = 0.0

    avg_loss = 0
    avg_acc = 0
    avg_loss_val = 0
    avg_acc_val = 0

    train_batches = len(dataloaders[TRAIN])
    val_batches = len(dataloaders[VAL])
    torch.cuda.empty_cache()

    for epoch in range(num_epochs):
        print("Epoch {}/{}".format(epoch + 1, num_epochs))
        print('-' * 10)

        loss_train = 0
        loss_val = 0
        acc_train = 0
        acc_val = 0

        vgg.train(True)

        for i, data in enumerate(dataloaders[TRAIN]):
            if i % 5 == 0:
                torch.cuda.empty_cache()
#                 print("\rTraining batch {}/{}".format(i, train_batches / 2),
end='', flush=True)

            # Use half training dataset
#             if i >= train_batches / 2:
#                 break

            inputs, labels = data

            if use_gpu:
                inputs, labels = Variable(inputs.cuda()), Variable(labels.cuda
())
            else:
                inputs, labels = Variable(inputs), Variable(labels)

            optimizer.zero_grad()

            outputs = vgg(inputs)
            _, preds = torch.max(outputs.data, 1)
            loss = criterion(outputs, labels)

            loss.backward()
            optimizer.step()

#             print("preds: ", preds)
#             print("labels.data: ", labels.data)
#             print(preds == labels.data)
#             print()

            loss_train += loss.data
            acc_train += torch.sum(preds == labels.data)

            del inputs, labels, outputs, preds
            torch.cuda.empty_cache()


        #print()
        # * 2 as we only used half of the dataset
        avg_loss = loss_train  / dataset_sizes[TRAIN]
        avg_acc = acc_train.item()  / dataset_sizes[TRAIN]

        vgg.train(False)
        vgg.eval()

        for i, data in enumerate(dataloaders[VAL]):
#             if i % 10 == 0:
#                 print("\rValidation batch {}/{}".format(i, val_batches), end
='', flush=True)

            inputs, labels = data

            if use_gpu:
                inputs, labels = Variable(inputs.cuda(), volatile=True), Varia
ble(labels.cuda(), volatile=True)
            else:
                inputs, labels = Variable(inputs, volatile=True), Variable(lab
els, volatile=True)

            optimizer.zero_grad()

            outputs = vgg(inputs)

            _, preds = torch.max(outputs.data, 1)
            loss = criterion(outputs, labels)

            loss_val += loss.data
            acc_val += torch.sum(preds == labels.data)

            del inputs, labels, outputs, preds
            torch.cuda.empty_cache()


        avg_loss_val = loss_val / dataset_sizes[VAL]
        avg_acc_val = acc_val.item() / dataset_sizes[VAL]

        print()
        print("Epoch {} result: ".format(epoch + 1))
        print("Avg loss (train): {:.4f}".format(avg_loss))
        print("Avg acc (train): {:.4f}".format(avg_acc))
        print("Avg loss (val): {:.4f}".format(avg_loss_val))
        print("Avg acc (val): {:.4f}".format(avg_acc_val))
        print('-' * 10)
#         print("avg_acc_val", avg_acc_val)
        print()
        if avg_acc_val > best_acc:
            best_acc = avg_acc_val
            best_model_wts = copy.deepcopy(vgg.state_dict())

    elapsed_time = time.time() - since
    print()
    print("Training completed in {:.0f}m {:.0f}s".format(elapsed_time // 60, elapsed_time % 60))
    print("Best acc: {:.4f}".format(best_acc))

    vgg.load_state_dict(best_model_wts)
    if(save):
      torch.save(best_model_wts, save_filename)
    return vgg
```

In [6]:

```
torch.cuda.empty_cache()
gpu_dtype = torch.cuda.FloatTensor

def get_vgg19(num_classes):
    net = models.vgg19_bn(pretrained=False)
    net.classifier = nn.Sequential(
        nn.Linear(25088, 4096),
        nn.ReLU(True),
        #nn.BatchNorm1d(4096),
        nn.Dropout(),
        nn.Linear(4096, 2048),
        nn.ReLU(True),
        #nn.BatchNorm1d(2048),
        nn.Dropout(),
        nn.Linear(2048, num_classes),
    )
    return net.type(gpu_dtype)

vgg19 = get_vgg19(6) #this fixes the issue where vgg outputs 1000 classifiers
 and now outputs n classifiers in get_vgg(n)
vgg19.cuda()

loss_fn = torch.nn.CrossEntropyLoss()
optimizer = optim.SGD(vgg19.parameters(), lr=0.005, momentum=0.9)
exp_lr_scheduler = lr_scheduler.StepLR(optimizer, step_size=7, gamma=0.1)

vgg19 = train_model(vgg19, loss_fn, optimizer, exp_lr_scheduler, num_epochs=2)
#saved version // remember to change the filename and follow naming scheme
#vgg19 = train_model(vgg19, loss_fn, optimizer, exp_lr_scheduler, num_epochs=1
25, save=True, save_filename='VGG16_bn_125e_adam05.pt')
```

```
Epoch 1/2
----------

/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:78: UserWarning:
volatile was removed and now has no effect. Use `with torch.no_grad():` inste
ad.

Epoch 1 result:
Avg loss (train): 0.1492
Avg acc (train): 0.3537
Avg loss (val): 0.1118
Avg acc (val): 0.3840
----------

Epoch 2/2
----------

Epoch 2 result:
Avg loss (train): 0.1484
Avg acc (train): 0.3367
Avg loss (val): 0.1436
Avg acc (val): 0.3920
----------


Training completed in 1m 21s
Best acc: 0.3920
```

In [7]:

```
def check_accuracy(model, loader):
#     if loader.dataset.train:
#         print('Checking accuracy on validation set')
#     else:
#         print('Checking accuracy on test set')
    num_correct = 0
    num_samples = 0
    model.train(False)
    model.eval() # Put the model in test mode (the opposite of model.train(),
 essentially)
    for x, y in loader:
        x_var = Variable(x.type(gpu_dtype), volatile=True)

        scores = model(x_var)
        _, preds = scores.data.cpu().max(1)
        print(preds)
        num_correct += (preds == y).sum()
        num_samples += preds.size(0)
    acc = float(num_correct) / num_samples
    print('Got %d / %d correct (%.2f)' % (num_correct, num_samples, 100 * acc
))


check_accuracy(vgg19, dataloaders[TRAIN])
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:11: UserWarning:
volatile was removed and now has no effect. Use `with torch.no_grad():` inste
ad.
  # This is added back by InteractiveShellApp.init_path()
```

```
tensor([5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5])
tensor([5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5])
tensor([5, 5, 5, 3, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5])
tensor([5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5])
tensor([5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5])
tensor([5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5])
tensor([5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5])
tensor([5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 3, 5])
tensor([5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5])
tensor([5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5])
tensor([5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5])
tensor([5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5])
tensor([5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5])
tensor([5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5])
tensor([5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5])
tensor([5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5])
tensor([5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5])
tensor([5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5])
tensor([5, 5, 5, 5, 5, 5])
Got 96 / 294 correct (32.65)
```

```
In [8]: check_accuracy(vgg19, dataloaders[VAL])
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:11: UserWarning:
volatile was removed and now has no effect. Use `with torch.no_grad():` inste
ad.
  # This is added back by InteractiveShellApp.init_path()

tensor([5, 5, 5, 5, 5, 5, 5, 3, 5, 5, 5, 5, 5, 5, 5, 5])
tensor([5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5])
tensor([5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5])
tensor([5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5])
tensor([5, 5, 5, 5, 5, 5, 3, 5, 5, 5, 5, 5, 5, 5, 5, 5])
tensor([5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5])
tensor([5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5])
tensor([5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5])
Got 49 / 125 correct (39.20)
```

```
In [9]: check_accuracy(vgg19, dataloaders[TEST])
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:11: UserWarning:
volatile was removed and now has no effect. Use `with torch.no_grad():` inste
ad.
  # This is added back by InteractiveShellApp.init_path()

tensor([5, 5, 5, 3, 5, 5, 5, 5, 5, 5, 3, 5, 5, 5, 5, 5])
tensor([5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5])
tensor([5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5])
tensor([5, 5, 5, 5, 5, 5, 3, 5, 5, 5, 5, 5, 5, 5, 5, 5])
tensor([5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5])
tensor([5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5])
tensor([5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5])
tensor([5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5])
Got 47 / 125 correct (37.60)
```